INTEL CORPORATION
5200 N.E. Elam Young Parkway
Hillsboro, Oregon 97124-9987
(503) 681-8080

intel®

Dear Customer:

Thank you for purchasing Intel's UNIX Release 4.0 Platform Manual Set. The Platform manuals include the complete set of UNIX Release 4.0 user- and administrator-level documentation.

The product you've received includes the following manuals:

User's Guide Vol. 1
User's Guide Vol. 2
User's Reference Vol. 2
System Administrator's Guide
System Administrator's Reference Vol. 1
System Administrator's Reference Vol. 2
Advanced Administration Vol. 1
Advanced Administration Vol. 2
Advanced Administration Vol. 3
Network Users & Administrators Guide Vol.1
Network Users & Administrators Guide Vol. 2
OPEN LOOK User's Guide
Migration Guide
Tips and Troubleshooting

Please call the appropriate number from the list below if any of these items are missing.

U.S.	800-INTEL4U
U.K.	(0) 793 / 641469
France	(16) 1 / 30.57.70.00
Germany	(0) 89 / 903-2025
Netherlands	(0) 10 / 4.07.11.30
Italy	(0) 2 / 892.00950
Israel	(0) 3 / 548-3222
Sweden	(0) 8 / 825416
Finland	(9) 0 / 544644
Spain	(9) 1 / 308-2552
Switzerland	call Germany
Denmark	call Sweden

Document Number	Sheet	Rev
466563	1	A

Norway	call Sweden
Other countries	call nearest facility

A product registration card is included in this package. Fill out the card, and mail the card to Intel using the mailing label for the Intel office nearest you.

Contact your Intel salesperson or Intel distributor for information on other Intel UNIX products.

Thanks for choosing Intel.

intel®

product release notes

UNIX* System V Release 4.0

Platform Manual Set Release Notes

DOCUMENT	SHEET	REV
466477	1	A

CONTENTS

1. Introduction

These Release Notes support the UNIX Release 4.0 Platform Manual Set. They include changes you should make to the manuals in the Platform documentation set. The changes are grouped by manual. This document also includes man pages that are missing from the manuals.

2. User's Reference Manual

Make the following changes to the *User's Reference Manual*:

- `sh`(1) man page: In the SYNOPSIS field, the line

  `rsh [-acefhiknprstuvx]` `[`*`args`*`]`

  should be:

  `/usr/lib/rsh [-acefhiknprstuvx]` `[`*`args`*`]`

- Missing man pages:

– `dfspace`(1)	
– `dos(1)`	See Section 7.
– `getclk`(1)	
– `tapecntl`(1)	See Section 7.
– `uugetty`(1C)	
– `newvt`(1M)	See Section 7.
– `vtlmgr`(1M)	See Section 7.

3. System Administrator's Guide

Make the following changes to the *System Administrator's Guide*:

- Chapter 1, Page 1-2: Ignore the last paragraph, it is irrelevant.
- Chapter 4, Pages 4-55 through 4-57: Ignore all text in the section titled "Using an Alternate or Remote Console".
- Chapter 9: The seventh paragraph refers to the *SCSI Operations Manual* and the *SCSI Installation Manual*. These manuals do not exist. Instead, see the *System Administrator's Guide* and the *System V Release 4.0 Installation Guide* for information.

4. System Administrator's Reference Manual

Make the following changes to the *System Administrator's Reference Manual*:

- The Table of Contents: Contains entries for the commands `hosts.equiv`(4), `.rhosts`(4), `hosts`(4), and `inetd.conf`(4). These commands are actually in the *Network User's and Administrator's Guide.*
- `ttymon`(1M) man page: The command syntax shown is incorrect. Use the syntax shown by the usage message. Get the message by typing **`ttymon`**.
- `archives`(4) man page: Values for the `t_typeflag` missing. The missing values are:

`'0'`	regular file
`'1'`	hard link
`'2'`	symbolic link
`'3'`	character special file
`'4'`	block special file
`'5'`	directory
`'6'`	FIFO

- `inittab`(4) man page: The DESCRIPTION section says the file is in `/sbin/inittab`. It's really in `/etc/inittab`.
- `vfstab`(4) man page: In the DESCRIPTION section, the reference to the `etc/vfstab` file should be `/etc/vfstab`.
- `iconv`(5) man page: In the FILES section, the files listed in the directory `/usr/lib/iconv` are really in `/usr/lib/locale`.
- Missing man pages:

– `evgainit`(1)	See Section 7.
– `fdisk`(1M)	See Section 7.
– `format`(1M)	See Section 7.
– `partsize`(1M)	
– `vtgetty`(1M)	See Section 7.
– `prototype`(4)	
– `termio`(7)	See Section 7.

5. Network User's and Administrator's Guide

Make the following changes in the *Network User's and Administrator's Guide*:

- `rsh`(1) man page: In the EXAMPLES section, the second example should be:

  rsh lizard cat lizard.file ">>" another.lizard.file

  appends the file `lizard.file` on the machine called "lizard" to the file `another.lizard.file` which also resides on the machine called "lizard".

- `telnet`(1) man page: The z option says `sh`(1) supports job control. It does not.

6. OPEN LOOK™ GUI User's Guide

Make the following changes to the *OPEN LOOK GUI User's Guide*:

- Page D-1: In the BNF syntax (shown in the second paragraph), a | (pipe) is missing between the words "menu" and "on" on the second line of the notation. The line should read: **`stmtm = menu | on`**
- `oladduser`(1) man page: Says that if there is an `.Xdefaults` file in the user's home directory, it renames it to `.Xdefaults.old`. The file is really renamed to `.Xdefaults-old`.
- Networking section fails to mention that `laitcp` and `iso-tp4` are also supported.

7. Man Pages

The following man pages were left out of the reference manuals and are included here. They belong in the following manuals:

`dos` (1)	*User's Reference Manual*
`evgainit`(1)	*System Administrator's Reference Manual*
`fdisk`(1M)	*System Administrator's Reference Manual*
`format`(1M)	*System Administrator's Reference Manual*
`newvt`(1M)	*User's Reference Manual*
`tapecntl`(1)	*User's Reference Manual*
`vtgetty`(1M)	*System Administrator's Reference Manual*
`vtlmgr`(1M)	*User's Reference Manual*
`x286emul`(1)	*Programmer's Reference Manual*
`termio`(7)	*System Administrator's Reference Manual*

NAME

`dos`: `doscat`, `doscp`, `dosdir`, `dosformat`, `dosmkdir`, `dosls`, `dosrm`, `dosrmdir` – access and manipulate DOS files

SYNOPSIS

`doscat` [`-r` | `-m`] *file* ...

`doscp` [`-r` | `-m`] *file1 file2*

`doscp` [`-r` | `-m`] *file* ... *directory*

`dosdir` *directory*

`dosformat` [`-fqv`] *drive*

`dosls` *directory* ...

`dosmkdir` *directory* ...

`dosrm` *file* ...

`dosrmdir` *directory* ...

DESCRIPTION

The dos commands allow access to files and directories on a DOS hard disk partition or diskette. The DOS partition must be bootable, although not active.

Below is a description of the dos commands:

`doscat`	Copies one or more DOS files to the standard output. If `-r` is given, the files are copied without newline conversions. If `-m` is given, the files are copied with newline conversions.
`doscp`	Copies files from/to a DOS diskette or a DOS hard disk partition to/from a UNIX file system. If *file1* and *file2* are given, *file1* is copied to *file2*. If a *directory* is given, one or more *files* are copied to that directory. If `-r` is given, the files are copied without new line conversions. If `-m` is given, the files are copied with newline conversions.
`dosdir`	Lists DOS files in the standard DOS style directory format. (See the DOS `DIR` command.)
`dosformat`	Creates a DOS 2.0 formatted diskette. It cannot be used to format a hard disk partition. The drive may be specified in either DOS drive convention, using the default file `/etc/default/msdos`, or using the UNIX special file name. The `-f` option suppresses the interactive feature. The `-q` (quiet) option is used to suppress information normally displayed during `dosformat`, but it does not suppress the interactive feature. The `-v` option prompts the user for a volume label after the diskette has been formatted. The maximum size of the volume label is 11 characters.
`dosls`	Lists DOS directories and files in a UNIX system style format [see `ls`(1)].

`dosrm`	Removes DOS files.
`dosmkdir`	Creates DOS directories.
`dosrmdir`	Deletes DOS directories.

The *file* and *directory* arguments for DOS files and directories have the form:

device:*name*

where *device* is a UNIX system path name for the special device file containing the DOS disk, and *name* is a path name to a file or directory on the DOS disk. The two components are separated by a colon (:). For example, the argument:

```
/dev/rdsk/f0t:/src/file.c
```

specifies the DOS file `file.asm` in the directory `/src` on diskette `/dev/rdsk/fd0t`. Note that slashes (and not backslashes) are used as file name separators for DOS path names. Arguments without a *device*: are assumed to be UNIX files.

For convenience, the user-configurable default file `/etc/default/msdos` can define DOS drive names to be used in place of the special device file path names. It may contain the following lines:

```
A=/dev/rdsk/f0t
C=/dev/rdsk/f1t
D=/dev/rdsk/0s5
```

The drive letter "`A`" may be used in place of special device file path name `/dev/rdsk/f0t` when referencing DOS files (see "Examples" below). The drive letter "`B`" or "`D`" refer to the DOS partition on the first or second hard disk.

The commands operate on the following types of disks:

DOS partitions on a hard disk
5-1/4 inch DOS
3-1/2 inch DOS
8, 9, 15, or 18 sectors per track
40 tracks per side
1 or 2 sides
DOS versions 1.0, 2.0, or 3.0

In the case of **doscp**, certain name conversions can be performed when copying a UNIX system file. File names with a base name longer than eight characters are truncated. Filename extensions (the part of the name following the separating period) longer than three characters are truncated. For example, the file 123456789.12345 becomes 12345678.123. A message informs the user that the name has been changed and the altered name is displayed. File names containing illegal DOS characters are stripped when writing to the DOS format. A message informs the user that characters have been removed and displays the name as written.

All DOS text files use a carriage-return/linefeed combination, CR-LF, to indicate a newline. UNIX system text files use a single newline LF character. When the **doscat** and **doscp** commands transfer DOS text files to UNIX system text files, they automatically strip the CR. When text files are transferred to DOS, the commands insert a CR before each LF character.

Under some circumstances, the automatic newline conversions do not occur. The `-m` option may be used to ensure the newline conversion. The `-r` option can be used to override the automatic conversion and force the command to perform a true byte copy regardless of file type.

EXAMPLES

```
doscat /dev/rdsk/f0t:tmp/output.1
doscat /tmp/f1 /tmp/f2/A:prog/output.1

dosdir /dev/rdsk/f0t:/prog
dosdir /D:/prog

doscp /mine/file.out/dev/rdsk/f0t:/mine/file.2
doscp /tmp/f1 /tmp/f2 D:

dosformat /dev/rdsk/f0d8dt

dosls /dev/rdsk:/src
dosls B:

dosmkdir /dev/fd0:/usr/docs

dosrm /dev/rdsk:/docs/memo.txt
dosrm /A:/docs/memo1.txt

dosrmdir /dev/rdsk:/usr/docs
```

FILES

`/etc/default/msdos`	Default information
`/dev/rdsk/f0t`	Floppy disk devices
`/dev/rdsk/0s5`	Hard disk devices

BUGS

Problems may occur when you type

```
doscp -m a:DOS_file UNIX_file
```

or

```
doscp -m UNIX_file a:DOS _file.
```

The space for the carriage returns added to each line is not allocated and the final part of the file is not written out to the DOS file system. Reading the file back with:

```
doscp -m a:DOS_file UNIX_file
```

results in an error message of `Sector not found.`

This will be fixed in the next release of the command.

Workaround: Use the following procedure:

```
doscp -m UNIX_file UNIX_tmp_file
doscp UNIX_tmp_file a:DOS_file
```

The file can be read without error with the normal command/procedure:

```
        doscp -m a:DOS_file UNIX_file
```

SEE ALSO

assign(C), dtype(C). See your MS-DOS Documentation.

NOTES

It is not possible to refer to DOS directories with wild card specifications. The programs mentioned above cooperate among themselves so no two programs will access the same DOS disk. If a process attempts to access a device almost in use, it displays the error message, "Device Busy", and exits with and exit code of 1.

The device argument to dosformat must be specific. For example, use /dev/rdsk/f0d9dt not /dev/rdsk/f0t or t:.

NAME

evgainit – Extended VGA keyboard/display driver initialization

SYNOPSIS

`evgainit` *card-type*

DESCRIPTION

`evgainit` is used to initialize the keyboard/display driver (see the `keyboard(7)` manual page) if extended VGA graphics modes are being used on certain video cards.

The keyboard/display driver provides the interface to the video card. `evgainit` informs the keyboard/display driver which video card is installed and should be rerun each time the system is booted.

In many cases the keyboard/display driver can determine which card is being used and therefore this command need not be run. For example, you don't need to run `evgainit` for the following cards:

AT&T VDC 400, VDC 600, or VDC 750.

Any card that doesn't have extended VGA capability (i.e. 800x600 pixels).

Any card that is only VGA (640x480 pixels) or EGA (640x350 pixels).

Any extended VGA cards (listed below) that will not be set to graphics modes with resolutions greater than 640x480 pixels.

`evgainit` must be run, however, for the following cards before attempting to use resolutions greater than 640x480 pixels. The following list shows the *card-type* argument value that should be used for each video card:

card-type	Video Card(s)
`vega`	Video 7 800x600, Video 7 VEGA VGA Adaptor
`stbga`	STB VGA Extra/EM, Extra/EM-16
`sigma/h`	SIGMA VGA/H
`pvga1a`	Paradise PVGA1A –
`dell`	Dell VGA
`vram`	Video 7 VRAM VGA
`orvga`	Orchid Designer VGA, Designer 800 VGA, ProDesigner VGA
`orvgani`	Orchid Designer, ProDesigner VGA (non-interlaced)
`tvga`	Tseng Labs
`tvgani`	Tseng Labs (non-interlaced)
`gvga`	Genoa Super VGA
`pega`	Paradise PEGA2
`gega`	Genoa EGA
`fastwrite`	Video 7 FastWrite VGA
`won`	ATI VGA Wonder

The command can only be run with super user privileges.

EXAMPLES

For an STB Extra/EM-16 video card, `evgainit` should be invoked as:

```
    evgainit stb
```

This command could be run automatically from the `inittab` file (see the inittab(4) manual page) or could be run by super user after each system reboot.

SEE ALSO

`keyboard`(7), `console`(7), `inittab`(4).
"Video Interface" in the *Integrated Software Developer's Guide.*

NAME

`fdisk` – create or modify hard disk partition table

SYNOPSIS

`fdisk` [*argument*]

DESCRIPTION

This command is used to create and modify the partition table that is put in the first sector of the hard disk. This table is used by DOS and by the first-stage bootstrap to identify parts of the disk reserved for different operating systems, and to identify the partition containing the second-stage bootstrap (the active partition). The optional argument can be used to specify the raw device associated with the hard disk; the default value is `/dev/rdsk/0s0` for integral disks. For SCSI disks, there is no default value. However if the default on your system is set to `0s0`, then it is linked to `/dev/rdsk/c0t0d0s0`. If the default is set to `1s0`, then it is linked to `/dev/rdsk/c0t1d0s0`.

The program displays the partition table as it exists on the disk, and then presents a menu allowing the user to modify the table. The menu, questions, warnings, and error messages are intended to be self-explanatory.

If there is no partition table on the disk, the user is given the option of creating a default partitioning or specifying the initial table values. The default partitioning allows 10% of the disk for MS-DOS and 90% for the UNIX System, and makes the UNIX System partition active. In either case, when the initial table is created, `fdisk` also writes out the first-stage bootstrap code [see hd(7)] along with the partition table. After the initial table is created, only the table is changed; the bootstrap is not modified.

Menu Options

The following are the menu options given by the `fdisk` program:

`Create a partition`

This option allows the user to create a new partition. The maximum number of partitions is 4. The program will ask for the type of the partition (MS-DOS, UNIX System, or other). It will then ask for the size of the partition as a percentage of the disk. The user may also enter the letter `c` at this point, in which case the program will ask for the starting cylinder number and size of the partition in cylinders. If a `c` is not entered, the program will determine the starting cylinder number where the partition will fit. In either case, if the partition would overlap an existing partition, or will not fit, a message is displayed and the program returns to the original menu.

`Change Active (Boot from) partition`

This option allows the user to specify the partition where the first-stage bootstrap will look for the second-stage bootstrap, otherwise known as the active partition.

`Delete a partition`

This option allows the user to delete a previously created partition. Note that this will destroy all data in that partition.

`Exit` This option writes the new version of the table created during this session with `fdisk` out to the hard disk, and exits the program.

`Cancel`
This option exits without modifying the partition table.

DIAGNOSTICS

Most messages will be self-explanatory. The following may appear immediately after starting the program:

`fdisk: cannot open <device>`
This indicates that the device name argument is not valid.

`fdisk: unable to get device parameters for device <device>`
This indicates a problem with the configuration of the hard disk, or an error in the hard disk driver.

`fdisk: error reading partition table`
This indicates that some error occurred when trying initially to read the hard disk. This could be a problem with the hard disk controller or driver, or with the configuration of the hard disk.

This message may appear after selecting the `Exit` option from the menu.

`fdisk: error writing boot record`
This indicates that some error occurred when trying to write the new partition table out to the hard disk. This could be a problem with the hard disk controller, the disk itself, the driver, or the configuration of the hard disk.

FILES

`/dev/rdsk/0s0` for integral disks

`/dev/rdsk/c?t?d?s0` for SCSI disks

SEE ALSO

`mkpart`(1M), `disk`(7), `hd`(7).

WARNINGS

Compatible with MS-DOS Versions 3.2, 3.3, and 4.0. Partitions set up using the MS-DOS 4.0 `fdisk` command that are greater than 32 MB will appear in the UNIX System display as "other". Partitions created with MS-DOS that are less than 32 MB will appear correctly as DOS partitions.

The DOS 4.01 `fdisk` program assumes it can store diagnostic information in cylinder 1020 on the hard disk. If a UNIX System partition is created that uses cylinder 1020, DOS 4.01 `fdisk` will be unable to create a DOS partition. Therefore, the user must either create the UNIX System partition at the front of the disk so that cylinder 1020 is not used, or create the DOS partition using the UNIX System `fdisk` (not DOS `fdisk`) and never delete it.

When setting up a DOS 4.01 partition on the hard disk to co-reside with a UNIX partition that has already been set up, DO NOT allow `fdisk` to create the largest possible partition and make it active (as the `fdisk` prompt requests). Instead, the user should manually set it up to line up against the UNIX partition. Note that this applies to when the user boots DOS 4.01 from floppy disk (not from within

UNIX) and runs fdisk.

NAME

`format` – format floppy disk tracks

SYNOPSIS

`/bin/format` [`-vVE`] [`-f` *first*] [`-l` *last*] [`-i` *interleave*] *device* [`t`]

DESCRIPTION

The `format` command formats floppy disks. Unless otherwise specified, formatting starts at track 0 and continues until an error is returned at the end of a partition.

The `-f` and `-l` options specify the first and last track to be formatted. The default interleave of 2 may be modified by using the `-i` option. *device* must specify a raw (character) floppy device. The `t` indicates the entire disk. Absence of this letter indicates that the first track of the diskette cannot be accessed.

`-v`	verbose.
`-V`	verify. After tracks are formatted, a random sector is chosen and a write of test data is done into it. The sector is then read back and a comparison is made.
`-E`	exhaustive verify. Every sector is verified by write/read/compare.

FILES

`/dev/rdsk/*`	raw device for partition to be formatted

SEE ALSO

`mkpart`(1M), `fd`(7).

NAME

newvt – opens virtual terminals.

SYNOPSIS

newvt [-e*prog*] [-n *vt_number*]

DESCRIPTION

The newvt command is a program invoked by the user to open a new virtual terminal (vt). The newly opened vt will inherit the user environment that existed when newvt was invoked.

-e Specifies a program (*prog*) to execute in the new vt. Without the -e option, the program pointed to by the $SHELL environment variable will be started in the new vt. If $SHELL is NULL or points to a nonexecutable program, then /bin/sh will be invoked.

-n Specifies a particular vt (*vt_number*) to open. If the -n option is not specified, then the next available vt will be opened. vt will be activated and the user can manually close the vt. This will be repeated until all open vts are manually closed.

DIAGNOSTICS

The command will fail under the following conditions:

If an illegal option is specified.
If the device cannot be opened.
If newvt is invoked from a remote terminal.
If no vts are available (-n option not specified).
If the requested vt is not available (-n option specified).
If the requested vt cannot be opened.
If the specified command cannot be executed (-e option specified).
If the $SHELL program cannot be executed ($SHELL set and -e option not specified).
If /dev/vtmon cannot be opened.

SEE ALSO

vtlmgr(1M) in the *User's Reference Manual*
vtgetty(1M) in the *System Administrator's Reference Manual.*

NAME

`tapecntl` – tape control for tape device

SYNOPSIS

`tapecntl` [`-bluetrwv`] [`-f` *arg*] [`-p` *arg*] [`special`]

DESCRIPTION

`tapecntl` will send the optioned commands to the tape device driver sub-device `/dev/rmt/c0s0` for all options except the `-e` option (position), which will use sub-device `/dev/rmt/c0s0n` using the `ioctl` command function. Sub-device `/dev/rmt/c0s0` provides a rewind on close capability, while `/dev/rmt/c0s0n` allows for closing of the device without rewind. Error messages will be written to standard error. Special is the tape device, and it defaults to `/dev/rmt/c0s0n` if not specified.

Not all options are supported by all tape devices and tape device drivers.

The meaning of the options are:

`-b` block length limits
Reads block length limits from the tape device and displays them.

`-l` load tape
Loads the tape media to the tape device and positions the tape at BOT.

`-u` unload tape
Unloads the tape media from the tape device. Depending on the device, unloading may include ejecting the catridge.

`-e` erase tape
Erasing the tape causes the erase bar to be activated while moving the tape from end to end, causing all data tracks to be erased in a single pass over the tape.

`-t` retension tape
Retensioning the tape causes the tape to be moved from end to end, thereby repacking the tape with the proper tension across its length.

`-r` reset tape device
Reset of the tape device initializes the tape controller registers and positions the tape at the beginning of the tape mark (BOT).

`-w` rewind tape
Rewinding the tape will move the tape to the BOT.

`-v` set variable length block mode
Sets the tape device to read and write variable length blocks.

`-f`[*n*] set fixed length block mode
sets the tape device to read abd write in fixed length blocks of *n* bytes.

−p[*n*] position tape to "end of file" mark − *n*
Positioning the tape command requires an integer argument. Positioning the tape will move the tape forward relative to its current position to the end of the specified file mark. The positioning option used with an argument of zero will be ignored. Illegal or out-of-range value arguments to the positioning command will leave the tape positioned at the end of the last valid file mark.

Options may be used individually or strung together with selected options being executed sequentially from left to right in the command line.

FILES

```
/sbin/tapecntl
/dev/rmt/c0s0n
/dev/rmt/c0s0
```

NOTES

Exit codes and their meanings are as follows:

exit (1)	device function could not initiate properly due to misconnected cables or poorly inserted tape cartridge.
exit (2)	device function failed to complete properly due to unrecoverable error condition, either in the command setup or due to mechanical failure.
exit (3)	device function failed due to the cartridge being write protected or to the lack of written data on the tape.
exit (4)	device `/dev/rmt/c0s0n` or `/dev/rmt/c0s0` failed to open properly due to already being opened or claimed by another process.

NAME

`vtgetty` – sets terminal type, modes, speed, and line discipline.

SYNOPSIS

`/etc/vtgetty [-h] [-t` *timeout*`]` *line* [[*speed*[*type* [*linedisc*]]]

DESCRIPTION

The `vtgetty` command is a program invoked by `init`(1M). It is the second process in the series (`init-vtgetty-getty-login-shell`) that passes its arguments and executes `/etc/getty`. The `/etc/getty` process will ultimately connect a user with the UNIX system. `vtgetty` can be executed only by the super-user (a process with the user-ID of root).

The command options are identical to those of `getty`(1M).

Initially, `vtgetty` opens the device and determines if any virtual terminals (vts) are open for that device. If there are active vts, the user will be prompted to determine if the vts should be closed automatically or manually when the user logs out.

If the automatic option is selected, `vtgetty` will send the signals, `SIGHUP` and `SIGTERM`, to each open vt.

It will then wait 3 seconds and send a `SIGKILL` signal to the vts to ensure that all the vts are terminated.

If the manual closure option is selected, the highest numbered vt will be activated and the user can manually close the vt. This will be repeated until all open vts are manually closed.

DIAGNOSTICS

`vtgetty` will fail under the following conditions:

If there is no memory available.
If it cannot open the device it was given.
If it cannot convert from a file descriptor to a file pointer.
If it cannot get the file status [`stat`(2)] of the device it was given.
If an `ioctl`(2) call fails.

FILES

`/etc/gettydefs`

SEE ALSO

`getty`(1M), `init`(1M), `kill`(1M), `tty`(1M), `ioctl`(2), `stat`(2), `gettydefs`(4), `inittab`(4) in the *Programmer's Reference Manual.*

NAME

vtlmgr – monitors and opens virtual terminals.

SYNOPSIS

`vtlmgr [-k]`

DESCRIPTION

The `vtlmgr` command is a program invoked by the user (usually from within the user's `.profile`) that places itself in the background and monitors `/dev/vtmon` for signals from the keyboard/display driver to open new virtual terminals (vts). With `vtlmgr` executing in the background, new vts will be opened whenever the user enters the "hot-key" sequence (default is `alt-sysreq` followed by a function key) configured for creating vts.

Upon entering the hot-key sequence, the executable program pointed to by the `$SHELL` variable will be executed in the new vt. If `$SHELL` is NULL or pointing to a not executable program `/bin/sh` will be executed. The newly opened vt will inherit the environment in effect when the `vtlmgr` command was invoked.

A user may perform setup on each new VT as it is created by `vtlmgr` through the `.vtlrc` file. This file should be in the user's home directory. Its contents are a shell script that will be run by `/bin/sh` before the shell prompt is displayed. In this way it is similar to the user's `.profile` file. However, no environment variables may be set and exported to the shell for the VT since a different shell runs the *.vtlrc* shell script.

DIAGNOSTICS

Errors will be reported under the following conditions:

If an illegal option is specified.
If the device cannot be opened.
If the command is invoked from a remote terminal.
If `/dev/vtmon` cannot be opened.
If `$SHELL` is set and is not executable.
If `$SHELL` is not set and `/bin/sh` cannot be invoked.

SEE ALSO

`newvt(1M)` in the *User's Reference Manual*
`vtgetty`(1M), `keyboard`(7) in the *System Administrator's Reference Manual*

NAME

x286emul – emulate XENIX 80286

SYNOPSIS

`x286emul [arg . . .] prog286`

DESCRIPTION

`x286emul` is an emulator that allows programs from XENIX System V/286 Release 2.3 or SCO's XENIX System V/286 Release 2.3.2 on the Intel 80286 to run on the Intel 80386 processor under UNIX System V.

The UNIX system recognizes an attempt to exec(2) a 286 program, and automatically exec's the 286 emulator with the 286 program name as an additional argument. It is not necessary to specify the `x286emul` emulator on the command line. The 286 programs can be invoked using the same command format as on the XENIX System V/286.

`x286emul` reads the 286 program's text and data into memory and maps them through the LDT [via `sysi86`(2)] as 286 text and data segments. It also fills in the jam area, which is used by XENIX programs to do system calls and signal returns. `x286emul` starts the 286 program by jumping to its entry point.

When the 286 program attempts to do a system call, `x286emul` takes control. It does any conversions needed between the 286 system call and the equivalent 386 system call, and performs the 386 system call. The results are converted to the form the 286 program expects, and the 286 program is resumed.

The following are some of the differences between a program running on a 286 and a 286 program using `x286emul` on a 386:

> Attempts to unlink or write on the 286 program will fail on the 286 with `ETXTBSY`. Under `x286emul`, they will not fail.
>
> `ptrace`(2) is not supported under `x286emul`.
>
> The 286 program must be readable for the emulator to read it.

The emulator must have this name and be in `/bin` if it is to be automatically invoked when exec(2) is used on a 286 program.

NAME

termio — general terminal interface

SYNOPSIS

```
#include <termio.h>

ioctl(int fildes, int request, struct termio *arg);
ioctl(int fildes, int request, int arg);

#include <termios.h>

ioctl(int fildes, int request, struct termios *arg);
```

DESCRIPTION

System V supports a general interface for asynchronous communications ports that is hardware-independent. The user interface to this functionality is via function calls (the preferred interface) described in `termios`(2) or `ioctl` commands described in this section. This section also discusses the common features of the terminal subsystem which are relevant with both user interfaces.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open terminal files; they are opened by the system and become a user's standard input, output, and error files. The very first terminal file opened by the session leader, which is not already associated with a session, becomes the controlling terminal for that session. The controlling terminal plays a special role in handling quit and interrupt signals, as discussed below. The controlling terminal is inherited by a child process during a `fork`(2). A process can break this association by changing its session using `setsid`(2).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the character input buffers of the system become completely full, which is rare (e.g., if the number of characters in the line discipline buffer exceeds {`MAX_CANON`} and `IMAXBEL` [see below] is not set), or when the user has accumulated {`MAX_INPUT`} number of input characters that have not yet been read by some program. When the input limit is reached, all the characters saved in the buffer up to that point are thrown away without notice.

Session management (Job Control)

A control terminal will distinguish one of the process groups in the session associated with it to be the foreground process group. All other process groups in the session are designated as background process groups. This foreground process group plays a special role in handling signal-generating input characters, as discussed below. By default, when a controlling terminal is allocated, the controlling process's process group is assigned as foreground process group.

Background process groups in the controlling process's session are subject to a job control line discipline when they attempt to access their controlling terminal. Process groups can be sent signals that will cause them to stop, unless they have made other arrangements. An exception is made for members of orphaned process groups. These are process groups which do not have a member with a parent in another process group that is in the same session and therefore shares the same controlling terminal. When a member's orphaned process group attempts to access its controlling terminal, errors will be returned. since there is no process to continue it if it should stop.

If a member of a background process group attempts to read its controlling terminal, its process group will be sent a SIGTTIN signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding SIGTTIN, or is a member of an orphaned process group, the read will fail with errno set to EIO, and no signal will be sent.

If a member of a background process group attempts to write its controlling terminal and the TOSTOP bit is set in the c_lflag field, its process group will be sent a SIGTTOU signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding SIGTTOU, the write will succeed. If the process is not ignoring or holding SIGTTOU and is a member of an orphaned process group, the write will fail with errno set to EIO, and no signal will be sent.

If TOSTOP is set and a member of a background process group attempts to ioctl its controlling terminal, and that ioctl will modify terminal parameters (e.g., TCSETA, TCSETAW, TCSETAF, or TIOCSPGRP), its process group will be sent a SIGTTOU signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding SIGTTOU, the ioctl will succeed. If the process is not ignoring or holding SIGTTOU and is a member of an orphaned process group, the write will fail with errno set to EIO, and no signal will be sent.

Canonical mode input processing

Normally, terminal input is processed in units of lines. A line is delimited by a newline (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not necessary, however, to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. The ERASE character (by default, the character number #) erases the last character typed. The WERASE character (the character control-W) erases the last "word" typed in the current input line (but not any preceding spaces or tabs). A "word" is defined as a sequence of non-blank characters, with tabs counted as blanks. Neither ERASE nor WERASE will erase beyond the beginning of the line. The KILL character (by default, the character @) kills (deletes) the entire input line, and optionally outputs a newline character. All these characters operate on a key stroke basis, independent of any backspacing or tabbing that may have been done. The REPRINT character (the character control-R) prints a newline followed by all characters that have not been read. Reprinting also occurs automatically if characters that would normally be erased from the screen are fouled by program output. The characters are reprinted as if they were being echoed; consequencely, if ECHO is not set, they are not printed.

The ERASE and KILL characters may be entered literally by preceding them with the escape character (\ ). In this case, the escape character is not read. The erase and kill characters may be changed.

Non-canonical mode input processing

In non-canonical mode input processing, input characters are not assembled into lines, and erase and kill processing does not occur. The `MIN` and `TIME` values are used to determine how to process the characters received.

`MIN` represents the minimum number of characters that should be received when the read is satisfied (i.e., when the characters are returned to the user). `TIME` is a timer of 0.10-second granularity that is used to timeout bursty and short-term data transmissions. The four possible values for `MIN` and `TIME` and their interactions are described below.

Case A: `MIN` > 0, `TIME` > 0

In this case, `TIME` serves as an intercharacter timer and is activated after the first character is received. Since it is an intercharacter timer, it is reset after a character is received. The interaction between `MIN` and `TIME` is as follows: as soon as one character is received, the intercharacter timer is started. If `MIN` characters are received before the intercharacter timer expires (note that the timer is reset upon receipt of each character), the read is satisfied. If the timer expires before `MIN` characters are received, the characters received to that point are returned to the user. Note that if `TIME` expires, at least one character will be returned because the timer would not have been enabled unless a character was received. In this case (`MIN` > 0, `TIME` > 0), the read sleeps until the `MIN` and `TIME` mechanisms are activated by the receipt of the first character. If the number of characters read is less than the number of characters available, the timer is not reactivated and the subsequent read is satisfied immediately.

Case B: `MIN` > 0, `TIME` = 0

In this case, since the value of `TIME` is zero, the timer plays no role and only `MIN` is significant. A pending read is not satisfied until `MIN` characters are received (the pending read sleeps until `MIN` characters are received). A program that uses this case to read record based terminal I/O may block indefinitely in the read operation.

Case C: `MIN` = 0, `TIME` > 0

In this case, since `MIN` = 0, `TIME` no longer represents an intercharacter timer: it now serves as a read timer that is activated as soon as a `read` is done. A read is satisfied as soon as a single character is received or the read timer expires. Note that, in this case, if the timer expires, no character is returned. If the timer does not expire, the only way the read can be satisfied is if a character is received. In this case, the read will not block indefinitely waiting for a character; if no character is received within `TIME`*.10 seconds after the read is initiated, the read returns with zero characters.

Case D: `MIN` = 0, `TIME` = 0

In this case, return is immediate. The minimum of either the number of characters requested or the number of characters currently available is returned without waiting for more characters to be input.

Comparison of the different cases of MIN, TIME interaction

Some points to note about MIN and TIME:

1. In the following explanations, note that the interactions of MIN and TIME are not symmetric. For example, when MIN > 0 and TIME = 0, TIME has no effect. However, in the opposite case, where MIN = 0 and TIME > 0, both MIN and TIME play a role in that MIN is satisfied with the receipt of a single character.

2. Also note that in case A (MIN > 0, TIME > 0), TIME represents an intercharacter timer, whereas in case C (TIME = 0, TIME > 0), TIME represents a read timer.

These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B, where MIN > 0, exist to handle burst mode activity (e.g., file transfer programs), where a program would like to process at least MIN characters at a time. In case A, the intercharacter timer is activated by a user as a safety measure; in case B, the timer is turned off.

Cases C and D exist to handle single character, timed transfers. These cases are readily adaptable to screen-based applications that need to know if a character is present in the input queue before refreshing the screen. In case C, the read is timed, whereas in case D, it is not.

Another important note is that MIN is always just a minimum. It does not denote a record length. For example, if a program does a read of 20 bytes, MIN is 10, and 25 characters are present, then 20 characters will be returned to the user.

Writing characters

When one or more characters are written, they are transmitted to the terminal as soon as previously written characters have finished typing. Input characters are echoed as they are typed if echoing has been enabled. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue is drained down to some threshold, the program is resumed.

Special characters

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR	(Rubout or ASCII DEL) generates a SIGINT signal. SIGINT is sent to all frequent processes associated with the controlling terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed upon location. [See signal(5)].
QUIT	(CTRL-| or ASCII FS) generates a SIGQUIT signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called core) will be created in the current working directory.

ERASE	(#) erases the preceding character. It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.
WERASE	(CTRL-W or ASCII ETX) erases the preceding "word". It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.
KILL	(@) deletes the entire line, as delimited by a NL, EOF, EOL, or EOL2 character.
REPRINT	(CTRL-R or ASCII DC2) reprints all characters, preceded by a newline, that have not been read.
EOF	(CTRL-D or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a newline, and the EOF is discarded. Thus, if no characters are waiting (i.e., the EOF occurred at the beginning of a line) zero characters are passed back, which is the standard end-of-file indication. Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.
NL	(ASCII LF) is the normal line delimiter. It cannot be changed or escaped.
EOL	(ASCII NULL) is an additional line delimiter, like NL. It is not normally used.
EOL2	is another additional line delimiter.
SWTCH	(CTRL-Z or ASCII EM) is used only when `shl` layers is invoked.
SUSP	(CTRL-Z or ASCII SUB) generates a `SIGTSTP` signal. `SIGTSTP` stops all processes in the foreground process group for that terminal.
DSUSP	(CTRL-Y or ASCII EM) It generates a `SIGTSTP` signal as SUSP does, but the signal is sent when a process in the foreground process group attempts to read the DSUSP character, rather than when it is typed.
STOP	(CTRL-S or ASCII DC3) can be used to suspend output temporarily. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.
START	(CTRL-Q or ASCII DC1) is used to resume output. Output has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read.
DISCARD	(CTRL-O or ASCII SI) causes subsequent output to be discarded. Output is discarded until another DISCARD character is typed, more input arrives, or the condition is cleared by a program.
LNEXT	(CTRL-V or ASCII SYN) causes the special meaning of the next character to be ignored. This works for all the special characters mentioned above. It allows characters to be input that would otherwise be interpreted by the system (e.g. KILL, QUIT).

The character values for INTR, QUIT, ERASE, WERASE, KILL, REPRINT, EOF, EOL, EOL2, SWTCH, SUSP, DSUSP, STOP, START, DISCARD, and LNEXT may be changed to suit individual tastes. If the value of a special control character is _POSIX_VDISABLE (0), the function of that special control character is disabled. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done. Any of the special characters may be preceded by the LNEXT character, in which case no special function is done.

Modem disconnect

When a modem disconnect is detected, a `SIGHUP` signal is sent to the terminal's controlling process. Unless other arrangements have been made, these signals cause the process to terminate. If `SIGHUP` is ignored or caught, any subsequent read returns with an end-of-file indication until the terminal is closed.

If the controlling process is not in the foreground process group of the terminal, a `SIGTSTP` is sent to the terminal's foreground process group. Unless other arrangements have been made, these signals cause the processes to stop.

Processes in background process groups that attempt to access the controlling terminal after modem disconnect while the terminal is still allocated to the session will receive appropriate `SIGTTOU` and `SIGTTIN` signals. Unless other arrangements have been made, this signal causes the processes to stop.

The controlling terminal will remain in this state until it is reinitialized with a successful open by the controlling process, or deallocated by the controlling process.

Terminal parameters

The parameters that control the behavior of devices and modules providing the `termios` interface are specified by the `termios` structure defined by `<termios.h>`. Several `ioctl(2)` system calls that fetch or change these parameters use this structure that contains the following members:

```
tcflag_t   c_iflag;           /* input modes */
tcflag_t   c_oflag;           /* output modes */
tcflag_t   c_cflag;           /* control modes */
tcflag_t   c_lflag;           /* local modes */
cc_t       c_cc[NCCS];        /* control chars */
```

The special control characters are defined by the array c_cc. The symbolic name NCCS is the size of the control-character array and is also defined by `<termios.h>`. The relative positions, subscript names, and typical default values for each function are as follows:

0	VINTR	DEL
1	VQUIT	FS
2	VERSE	#
3	VKILL	@
4	VEOF	EOT
5	VEOL	NUL
6	VEOL2	NUL
7	VSWTCH	NUL
8	VSTRT	DC1
9	VSTOP	DC3

10	VSUSP	SUB
11	VDSUSP	EM
12	VREPRINT	DC2
13	VDISCRD	SI
14	VWERSE	ETB
15	VLNEXT	SYN
16-19	reserved	

Input modes

The c_iflag field describes the basic terminal input control:

IGNBRK	Ignore break condition.
BRKINT	Signal interrupt on break.
IGNPAR	Ignore characters with parity errors.
PARMRK	Mark parity errors.
INPCK	Enable input parity check.
ISTRIP	Strip character.
INLCR	Map NL to CR on input.
IGNCR	Ignore CR.
ICRNL	Map CR to NL on input.
IUCLC	Map upper-case to lower-case on input.
IXON	Enable start/stop output control.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IMAXBEL	Echo BEL on input line too long.

If IGNBRK is set, a break condition (a character framing error with data all zeros) detected on input is ignored, that is, not put on the input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the break condition shall flush the input and output queues and if the terminal is the controlling terminal of a foreground process group, the break condition generates a single SIGINT signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break condition is read as a single ASCII NULL character ('\0'), or if PARMRK is set, as '\377', '\0', '\0'.

If IGNPAR is set, a byte with framing or parity errors (other than break) is ignored.

If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than break) is given to the application as the three-character sequence: '\377', '\0', X, where X is the data of the byte received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of '\377' is given to the application as '\377', '\377'. If neither IGNPAR nor PARMRK is set, a framing or parity error (other than break) is given to the application as a single ASCII NULL character ('\0').

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors. Note that whether input parity checking is enabled or disabled is independent of whether parity detection is enabled or disabled. If parity detection is enabled but input parity checking is disabled, the hardware to which the terminal is connected will recognize the parity bit, but the terminal special file will not check whether this is set correctly or not.

If ISTRIP is set, valid input characters are first stripped to seven bits, otherwise all eight bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise, if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received upper case, alphabetic character is translated into the corresponding lower case character.

If IXON is set, start/stop output control is enabled. A received STOP character suspends output and a received START character restarts output. The STOP and START characters will not be read, but will merely perform flow control functions. If IXANY is set, any input character restarts output that has been suspended.

If IXOFF is set, the system transmits a STOP character when the input queue is nearly full, and a START character when enough input has been read so that the input queue is nearly empty again.

If IMAXBEL is set, the ASCII BEL character is echoed if the input stream overflows. Further input is not stored, but any input already present in the input stream is not disturbed. If IMAXBEL is not set, no BEL character is echoed, and all input present in the input queue is discarded if the input stream overflows.

The initial input control value is BRKINT, ICRNL, IXON, ISTRIP.

Output modes

The c_oflag field specifies the system treatment of output:

OPOST	Post-process output.
OLCUC	Map lower case to upper on output.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NULL.
NLDLY	Select newline delays:
NL0	
NL1	
CRDLY	Select carriage-return delays:
CR0	
CR1	
CR2	
CR3	
TABDLY	Select horizontal tab delays:
TAB0	or tab expansion:
TAB1	
TAB2	
TAB3	Expand tabs to spaces.
XTABS	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	
BS1	

VTDLY	Select vertical tab delays:
VT0	
VT1	
FFDLY	Select form feed delays:
FF0	
FF1	

If OPOST is set, output characters are post-processed as indicated by the remaining flags; otherwise, characters are transmitted without change.

If OLCUC is set, a lower case alphabetic character is transmitted as the corresponding upper case character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONRET is set, the NL character is assumed to do the carriage-return function; the column pointer is set to 0 and the delays specified for CR are used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases, a value of 0 indicates no delay. If OFILL is set, fill characters are transmitted for delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay. If OFDEL is set, the fill character is DEL; otherwise it is NULL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

Newline delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the newline delays. If OFILL is set, two fill characters are transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters, and type 2 transmits four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters are transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character is transmitted.

The actual delays depend on line speed and system load.

The initial output control value is OPOST, ONLCR, TAB3.

Control Modes

The c_cflag field describes the hardware control of the terminal:

CBAUD	Baud rate:
B0	Hang up
B50	50 baud
B75	75 baud

B110	110 baud
B134	134 baud
B150	150 baud
B200	200 baud
B300	300 baud
B600	600 baud
B1200	1200 baud
B1800	1800 baud
B2400	2400 baud
B4800	4800 baud
B9600	9600 baud
B19200	19200 baud
EXTA	External A
B38400	38400 baud
EXTB	External B

CSIZE	Character size:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits

CSTOPB	Send two stop bits, else one
CREAD	Enable receiver
PARENB	Parity enable
PARODD	Odd parity, else even
HUPCL	Hang up on last close
CLOCAL	Local line, else dial-up
CIBAUD	Input baud rate, if different from output rate
PAREXT	Extended parity for mark and space parity

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal is not asserted. Normally, this disconnects the line. If the CIBAUD bits are not zero, they specify the input baud rate, with the CBAUD bits specifying the output baud rate; otherwise, the output and input baud rates are both specified by the CBAUD bits. The values for the CIBAUD bits are the same as the values for the CBAUD bits, shifted left IBSHIFT bits. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used; otherwise, one stop bit is used. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled, and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set; otherwise, even parity is used.

If CREAD is set, the receiver is enabled. Otherwise, no characters are received.

If HUPCL is set, the line is disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal is not asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control; otherwise, modem control is assumed.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

Local modes

The c_lflag field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline provides the following:

ISIG	Enable signals.
ICANON	Canonical input (erase and kill processing).
XCASE	Canonical upper/lower presentation.
ECHO	Enable echo.
ECHOE	Echo erase character as BS-SP-BS.
ECHOK	Echo NL after kill character.
ECHONL	Echo NL.
NOFLSH	Disable flush after interrupt or quit.
TOSTOP	Send SIGTTOU for background output.
ECHOCTL	Echo control characters as ^*char*, delete as ^?.
ECHOPRT	Echo erase character as character erased.
ECHOKE	BS-SP-BS erase entire line on line kill.
FLUSHO	Output is being flushed.
PENDIN	Retype pending input at next read or input character.
IEXTEN	Enable extended (implementation-defined) functions.

If ISIG is set, each input character is checked against the special control characters INTR, QUIT, SWTCH, SUSP, STATUS, and DSUSP. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus, these special input functions are possible only if ISIG is set.

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, EOL, and EOL2. If ICANON is not set, read requests are satisfied directly from the input queue. A read is not satisfied until at least MIN characters have been received or the timeout value TIME has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an upper case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

```
for:   use:
`      \'
|      \!
~      \^
{      \(
}      \)
\      \\
```

For example, `A` is input as \a, \n as \\n, and \N as \\\n.

If `ECHO` is set, characters are echoed as received.

When `ICANON` is set, the following echo functions are possible.

1. If `ECHO` and `ECHOE` are set, and `ECHOPRT` is not set, the ERASE and WERASE characters are echoed as one or more ASCII BS SP BS, which clears the last character(s) from a CRT screen.
2. If `ECHO` and `ECHOPRT` are set, the first ERASE and WERASE character in a sequence echoes as a backslash (\), followed by the characters being erased. Subsequent ERASE and WERASE characters echo the characters being erased, in reverse order. The next non-erase character causes a slash (/) to be typed before it is echoed. `ECHOPRT` should be used for hard copy terminals.
3. If `ECHOKE` is set, the kill character is echoed by erasing each character on the line from the screen (using the mechanism selected by `ECHOE` and `ECHOPRT`).
4. If `ECHOK` is set, and `ECHOKE` is not set, the NL character is echoed after the kill character to emphasize that the line is deleted. Note that an escape character (\) or an LNEXT character preceding the erase or kill character removes any special function.
5. If `ECHONL` is set, the NL character is echoed even if `ECHO` is not set. This is useful for terminals set to local echo (so called half-duplex).

If `ECHOCTL` is set, all control characters (characters with codes between 0 and 37 octal) other than ASCII TAB, ASCII NL, the START character, and the STOP character, ASCII CR, and ASCII BS are echoed as `^X`, where `X` is the character given by adding 100 octal to the code of the control character (so that the character with octal code 1 is echoed as `^A`), and the ASCII DEL character, with code 177 octal, is echoed as `^?`.

If `NOFLSH` is set, the normal flush of the input and output queues associated with the INTR, QUIT, and SUSP characters is not done. This bit should be set when restarting system calls that read from or write to a terminal [see `sigaction`(2)].

If `TOSTOP` is set, the signal `SIGTTOU` is sent to a process that tries to write to its controlling terminal if it is not in the foreground process group for that terminal. This signal normally stops the process. Otherwise, the output generated by that process is output to the current output stream. Processes that are blocking or ignoring `SIGTTOU` signals are excepted and allowed to produce output, if any.

If FLUSHO is set, data written to the terminal is discarded. This bit is set when the FLUSH character is typed. A program can cancel the effect of typing the FLUSH character by clearing FLUSHO.

If PENDIN is set, any input that has not yet been read is reprinted when the next character arrives as input.

If IEXTEN is set, the following implementation-defined functions are enabled: special characters (WERASE, REPRINT, DISCARD, and LNEXT) and local flags (TOSTOP, ECHOCTL, ECHOPRT, ECHOKE, FLUSHO, and PENDIN).

The initial line-discipline control value is ISIG, ICANON, ECHO, ECHOK.

Minimum and Timeout

The MIN and TIME values are described above under *Non-canonical mode input processing*. The initial value of MIN is 1, and the initial value of TIME is 0.

Terminal size

The number of lines and columns on the terminal's display is specified in the winsize structure defined by <sys/termios.h> and includes the following members:

```
unsigned  short   ws_row;    /* rows, in characters */
unsigned  short   ws_col;    /* columns, in characters */
unsigned  short   ws_xpixel;/* horizontal size, in pixels */
unsigned  short   ws_ypixel;/* vertical size, in pixels */
```

Termio structure

The System V termio structure is used by some ioctls; it is defined by <sys/termio.h> and includes the following members:

```
unsigned  short  c_iflag;     /* input modes */
unsigned  short  c_oflag;     /* output modes */
unsigned  short  c_cflag;     /* control modes */
unsigned  short  c_lflag;     /* local modes */
char             c_line;      /* line discipline */
unsigned  char   c_cc[NCC];   /* control chars */
```

The special control characters are defined by the array c_cc. The symbolic name NCC is the size of the control-character array and is also defined by <termio.h>. The relative positions, subscript names, and typical default values for each function are as follows:

0	VINTR	DEL
1	VQUIT	FS
2	VERASE	#
3	VKILL	@
4	VEOF	EOT
5	VEOL	NUL
6	VEOL2	NUL
7	reserved	

The calls that use the termio structure only affect the flags and control characters that can be stored in the termio structure; all other flags and control characters are unaffected.

Modem lines

On special files representing serial ports, the modem control lines supported by the hardware can be read, and the modem status lines supported by the hardware can be changed. The following modem control and status lines may be supported by a device; they are defined by <sys/termios.h>:

TIOCM_LE	line enable
TIOCM_DTR	data terminal ready
TIOCM_RTS	request to send
TIOCM_ST	secondary transmit
TIOCM_SR	secondary receive
TIOCM_CTS	clear to send
TIOCM_CAR	carrier detect
TIOCM_RNG	ring
TIOCM_DSR	data set ready

TIOCM_CD is a synonym for TIOCM_CAR, and TIOCM_RI is a synonym for TIOCM_RNG. Not all of these are necessarily supported by any particular device; check the manual page for the device in question.

IOCTLS

The ioctls supported by devices and STREAMS modules providing the termios interface are listed below. Some calls may not be supported by all devices or modules. The functionality provided by these calls is also available through the preferred function call interface specified on termios(2).

TCGETS	The argument is a pointer to a termios structure. The current terminal parameters are fetched and stored into that structure.
TCSETS	The argument is a pointer to a termios structure. The current terminal parameters are set from the values stored in that structure. The change is immediate.
TCSETSW	The argument is a pointer to a termios structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.
TCSETSF	The argument is a pointer to a termios structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.
TCGETA	The argument is a pointer to a termio structure. The current terminal parameters are fetched, and those parameters that can be stored in a termio structure are stored into that structure.

TCSETA	The argument is a pointer to a termio structure. Those terminal parameters that can be stored in a termio structure are set from the values stored in that structure. The change is immediate.
TCSETAW	The argument is a pointer to a termio structure. Those terminal parameters that can be stored in a termio structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.
TCSETAF	The argument is a pointer to a termio structure. Those terminal parameters that can be stored in a termio structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.
TCSBRK	The argument is an int value. Wait for the output to drain. If the argument is 0, then send a break (zero valued bits for 0.25 seconds).
TCXONC	Start/stop control. The argument is an int value. If the argument is 0, suspend output; if 1, restart suspended output; if 2, suspend input; if 3, restart suspended input.
TCFLSH	The argument is an int value. If the argument is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.
TIOCGPGRP	The argument is a pointer to a pid_t. Set the value of that pid_t to the process group ID of the foreground process group associated with the terminal. See termios(2) for a description or TCGETPGRP.
TIOCSPGRP	The argument is a pointer to a pid_t. Associate the process group whose process group ID is specified by the value of that pid_t with the terminal. The new process group value must be in the range of valid process group ID values. Otherwise, the error EPERM is returned. See termios(2) for a description of TCSETPGRP.
TIOCGSID	The argument is a pointer to a pid_t. The session ID of the terminal is fetched and stored in the pid_t.
TIOCGWINSZ	The argument is a pointer to a winsize structure. The terminal driver's notion of the terminal size is stored into that structure.
TIOCSWINSZ	The argument is a pointer to a winsize structure. The terminal driver's notion of the terminal size is set from the values specified in that structure. If the new sizes are different from the old sizes, a SIGWINCH signal is set to the process group of the terminal.

`TIOCMBIS`	The argument is a pointer to an `int` whose value is a mask containing modem control lines to be turned on. The control lines whose bits are set in the argument are turned on; no other control lines are affected.
`TIOCMBIC`	The argument is a pointer to an `int` whose value is a mask containing modem control lines to be turned off. The control lines whose bits are set in the argument are turned off; no other control lines are affected.
`TIOCMGET`	The argument is a pointer to an `int`. The current state of the modem status lines is fetched and stored in the `int` pointed to by the argument.
`TIOCMSET`	The argument is a pointer to an `int` containing a new set of modem control lines. The modem control lines are turned on or off, depending on whether the bit for that mode is set or clear.

FILES

files in or under /dev

SEE ALSO

`fork`(2), `ioctl`(2), `setsid`(2), `signal`(2), `termios`(2), `streamio`(7).